

**数据结构课程设计报告**

题目：**银行业务模拟**

学 院 计算机学院

专 业 信息安全

年级班别 2017级（2）班

学 号 3117005278

学生姓名 林俊涵

指导教师 李小妹

编 号 19

成 绩 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2019 年 1 月

**报告：**

**报告内容：** □详细　 □完整　 □基本完整 □不完整

**设计方案：** □非常合理　 □合理　 □基本合理 □较差

**算**法实**现：** □全部实现　 □基本实现　 □部分实现 □实现较差

**测试样例：** □完备　 □比较完备　 □基本完备 □不完备

**文档格式：** □规范　 □比较规范　 □基本规范 □不规范

**答辩：**

□理解题目透彻，问题回答流利

□理解题目较透彻，回答问题基本正确

□部分理解题目，部分问题回答正确

□未能完全理解题目，答辩情况较差

**总评成绩：**

□优　　　□良　　　□中　　　□及格　　　□不及格

题目： 银行业务模拟（难度系数：1.3）

[运行环境]

VS

[问题描述]

客户业务分为两种。第一种是申请从银行得到一笔资金，即取款或借款。第二种是向银行投入一笔资金，即存款或还款。银行有两个服务窗口，相应地有两个队列。客户到达银行后先排第一个队。处理每个客户业务时，如果属于第一种，且申请额超出银行现存资金总额而得不到满足，则立刻排入第二个队等候，直至满足时才离开银行；否则业务处理完后立刻离开银行。每接待完一个第二种业务的客户，则顺序检查和处理（如果可能）第二个队列中的客户，对能满足的申请者予以满足，不能满足者重新排到第二个队列的队尾。注意，在此检查过程中，一旦银行资金总额少于或等于刚才第一个队列中最后一个客户(第二种业务)被接待之前的数额，或者本次已将第二个队列检查或处理了一遍，就停止检查(因为此时已不可能还有能满足者)转而继续接待第一个队列的客户。任何时刻都只开一个窗口。假设检查不需要时间。营业时间结束时所有客户立即离开银行。

写一个上述银行业务的事件驱动模拟系统，通过模拟方法求出客户在银行内逗留的平均时间。

[基本要求]

利用动态存储结构实现模拟，即利用**C**语言的动态分配函数malloc和free。

[选做内容]

自己实现动态数据类型。（已作）

1. 需求分析

计算来银行办理业务的用户平均逗留时间：

⑴ 在运行终端控制台进行信息输入：

输入信息包括(信息都为整数)：

1.银行拥有的款额 取值范围：[0, 正无穷]

2.银行营业时间 取值范围：[0, 1440]

3.用户最大交易金额 取值范围：(0, 银行拥有款额]

4.俩个用户到达时间的最大值 取值范围：(0, 银行营业时间]

5.用户最大交易时间 取值范围：(0, 银行营业时间]

⑵ 在运行终端控制台进行信息输出：

1.每个到达用户的到达时间，编号，要处理的金额，处理时间

2.事件表

3.总用户数

4.总逗留时间

5.平均逗留时间

⑶ 程序所能达到的功能；

1.输出每个用户的到达时间，编号，要处理的金额，处理时间

2.银行运营过程的全部事件（包括到达事件和离开事件）

3.计算银行运营过程的用户数，总逗留时间，平均逗留时间

⑷ 三组测试数据：

1.正常情况：

1.银行拥有的款额：10000

2.银行营业时间： 600

3.用户最大交易金额：2000

4.俩个用户到达时间的最大值：20

5.用户最大交易时间：15

应输出：25.48

2.极端的情况：两个到达事件之间的间隔时间很短，而客户的交易时间很长

1.银行拥有的款额：10000

2.银行营业时间： 600

3.用户最大交易金额：2000

4.俩个用户到达时间的最大值：5

5.用户最大交易时间：100

应输出：20.0

3.极端的情况：两个到达事件之间的间隔时间很长，而客户的交易时间很短

1.银行拥有的款额：10000

2.银行营业时间： 600

3.用户最大交易金额：2000

4.俩个用户到达时间的最大值：100

5.用户最大交易时间：5

应输出：88.38

2. 概要设计

typedef int Status;

#define OK 1

#define ERROR -1

typedef struct kehu\_event{

// 定义客户信息

int handel\_time; // 事件处理时间

int money; //需要处理的钱数目(负数表示取钱，正数表示存钱）

int No; //客户的序号

}ElemType;

typedef struct event{

/\*定义事件类型\*/

int index; /\*标志变量，1代表到达事件，其余代表离开事件\*/

int time; /\*事件发生时间\*/

int No; /\*触发该事件的客户序号\*/

}event;

typedef struct { /\*定义链表节点\*/

event data;

struct LNode \*next;

}LNode, \*Link;

typedef struct { /\*定义链表\*/

Link head, tail;

int len;

}\*LinkList;

typedef struct LQNode { /\*定义队列的节点\*/

ElemType data;

struct LQNode \*next;

}LQNode, \*QueuePtr;

typedef struct { /\*定义队列\*/

QueuePtr front;

QueuePtr rear;

int len;

}LQueue;

全局变量：

int total ;

/\*一天营业开始时银行拥有的款额为10000(元),并且默认取钱只能\*/

//int total\_time = 600;

/\*营业时间为600(分钟)\*/

int max\_money;

LQueue \*handel\_queue = NULL;

/\*定义第一个队列用于处理业务\*/

LQueue \*wait\_queue = NULL;

/\* 定义第二个队列用于存储等候的客户\*/

LinkList event\_link = NULL;

/\*定义一个事件表\*/

int nextTime = 0;

/\*定义一个变量作为下一个客户的随机到达时间\*/

int nexthandelTime = 0;

/\*定义一个变量作为正在排队的下一个客户处理时间\*/

int kehu\_NO = 1;

/\*客户的服务序号\*/

ElemType \*leave\_kehu = NULL;

/\*定义一个办理完业务要离开的客户变量，用于记录\*/

ElemType \*search\_kehu = NULL;

/\*定义一个用来存储待处理业务的客户变量\*/

3. 详细设计

链表操作接口实现：

LinkList InitLink() { /\*初始化链表操作\*/

LinkList L = (LinkList\*)malloc(sizeof(LNode));

if (L == NULL) return NULL;

L->head = NULL;

L->tail = NULL;

L->len = 0;

printf("链表初始化成功！！");

return L;

};

Status ClearLink(LinkList L) { /\*链表的清空操作\*/

if (L == NULL) return ERROR;

Link p = L->head;

Link k = L->head->next;

while (p != L->tail)

{

free(p);

p = k;

if(k!=L->tail) k = k->next;

}

free(p);

L->head = NULL;

L->tail = NULL;

L->len = 0;

printf("链表已经清空！！");

return OK;

}

Status DestoryLink(LinkList L) { /\*链表的销毁操作\*/

if (L == NULL) return ERROR;

ClearLink(L);

free(L);

printf("链表销毁完毕！！");

return OK;

}

Status InsertLink(LinkList L, event e) { /\*链表的尾部插入操作\*/

if (L == NULL) return ERROR;

Link p = (Link)malloc(sizeof(LNode));

if (p == NULL) return ERROR;

p->data = e;

p->next = NULL;

if (L->head == NULL) {

L->head = p;

L->tail = p;

L->len = 1;

}else{

L->tail->next = p;

L->tail = p;

L->len++;

}

//printf("节点插入成功！！");

return OK;

}

Status LinkTraverse(LinkList L) { /\*链表的遍历操作\*/

if (L == NULL) return ERROR;

Link p = L->head;

printf("\n遍历结果 :\n");

printf("事件类型(1代表到达,0代表离开）： 客户序号： 事件触发时间： ");

while (p != NULL) {

printf("\n %d %d %d", p->data.index, p->data.No, p->data.time );

p = p->next;

}

return OK;

}

队列操作接口实现：

LQueue\* InitQueue() { /\*构造一个空队列\*/

LQueue \*Q = (LQueue \*)malloc(sizeof(LQueue));

if (Q == NULL) return ERROR;

Q->front = NULL;

Q->rear = NULL;

Q->len = 0;

return Q;

}

Status EnQueue(LQueue \*Q, ElemType e) { //在队列的队尾插入e

LQNode \*p;

p = (LQNode\*)malloc(sizeof(LQNode));

if (p == NULL) return ERROR;

p->data = e;

p->next = NULL;

if (NULL == Q->front) {

Q->front = p;

Q->rear = p;

Q->len = 1;

}

else

{

Q->rear->next = p;

Q->rear = p;

Q->len++;

}

return OK;

}

Status DeQueue(LQueue \*Q, ElemType \*e) {

//是队列Q的队头出队并且返回到e

LQNode \*p;

if (NULL == Q->front) return ERROR;

p = Q->front;

\*e = p->data;

Q->front = p->next;

if (Q->rear == p) Q->rear = NULL;

free(p);

Q->len--;

return OK;

}

Status DestoryQueue(LQueue \*Q) {

/\*队列的销毁操作\*/

if (Q == NULL) return ERROR;

QueuePtr p=NULL, q=NULL;

p = Q->front;

while (p != Q->rear) {

q = p->next;

free(p);

p = q;

}

free(Q->rear);

return OK;

}

Status InsertQueue(LQueue \*Q, LQueue \*E) {

/\*将E队列插入到Q队列的前面，并将Q队列作为新的队列释放E\*/

if ( E == NULL) return ERROR;

if (Q == NULL) {

Q->front = E->front;

Q->rear = E->rear;

DestoryQueue(E);

}else

{

E->rear->next = Q->front;

Q->front = E->front;

}

return OK;

}

银行事件处理实现：

ElemType\* Arrial\_event(int i, int max\_timejiange, int max\_timejiaoyi) {

/\*用来处理新客户到达事件\*/

if (nextTime == i) {

printf("当前时间是:%d\n", i);

nextTime = nextTime + (rand() % max\_timejiange) + 1; //随机生成下一名客户到达时间，这里可以设置客户到达时间间隔的大小

ElemType \*p=NULL;

p = mymalloc(max\_timejiaoyi); //到达前面设置的到达时间时，随机生成刚到达用户的信息，包括客户编号，业务处理时间，业务操作钱的数目

EnQueue(handel\_queue, \*p);

event \*e = (event\*)malloc(sizeof(event)); //生成该到达事件的信息节点

e->index = 1;

e->No = p->No;

e->time = i;

InsertLink(event\_link, \*e); //将该到达事件的信息节点加入事件链表

if (nexthandelTime < i) { //如果当前时间已经超过了下一个用户操作时间，则将当前时间和该到达用户的时间相加作为下一个用户操作时间

nexthandelTime = i + p->handel\_time;

}

if (nexthandelTime == 0 && -(p->money)<= max\_money) {//第一个进入操作的用户情况

nexthandelTime = p->handel\_time;

}

printf("\n 客户编号：%d 客户操作需要时间：%d 客户需要处理的钱：%d\n\n\n", p->No, p->handel\_time, p->money);

return p;

}

}

Status handel\_event(int i) {

/\*处理队列事件\*/

if (nexthandelTime == i) { /\*到达下一个客户操作时间时\*/

DeQueue(handel\_queue, leave\_kehu); //将处理完的客户出队，并将客户信息存储在leave\_kehu变量中

int last\_money = total - leave\_kehu->money; //获取刚离开的客户操作前银行的钱数目

Leave\_event(leave\_kehu, i); //将离开事件的用户信息存入事件列表中

if (((leave\_kehu->money) > 0) && wait\_queue->front != NULL) {

/\*如果刚刚离开的用户是存钱或者还钱，并且 等待队列 不为空\*/

int index = 0; //设置标记变量，用以标记等待队列已经查看的客户数目

LQueue \*e = InitQueue(); //创造出一个可以被满足需求的等待用户组成的队列

while (total > last\_money) { //如果当前银行拥有的款额比之前多，进入循环

DeQueue(wait\_queue, search\_kehu); //将等待队列的队头用户出队

if (-(search\_kehu->money) <= total) {

/\*如果等待用户的需求可以得到满足\*/

EnQueue(e, \*search\_kehu); //将可满足用户加入等待用户队列

total = total + search\_kehu->money;

}

else {

EnQueue(wait\_queue, \*search\_kehu);

}

index++;

if (index >= wait\_queue->len) break;

}

InsertQueue(handel\_queue, e); //将可以满足要求的等待队列并入前面

//nexthandelTime = nexthandelTime + handel\_queue->front->data.handel\_time; //下一个处理用户时间更新

if (handel\_queue->front != NULL) {

/\*如果操作队列不为空\*/

while (-(handel\_queue->front->data.money) > total)

/\*如果无法满足要求\*/

{

DeQueue(handel\_queue, leave\_kehu);

EnQueue(wait\_queue, \*leave\_kehu);

}

nexthandelTime = handel\_queue->front->data.handel\_time + nexthandelTime; //下一位操作客户时间更新

total = total + handel\_queue->front->data.handel\_time; //银行金钱总额更新

}

}else {

/\*刚刚离开的用户是取钱或者借钱，或者 等待队列 为空\*/

if (handel\_queue->front != NULL) {

while (-(handel\_queue->front->data.money) > total)

{

/\*无法满足的用户进入等待队列\*/

DeQueue(handel\_queue, leave\_kehu);

EnQueue(wait\_queue, \*leave\_kehu);

}

if (handel\_queue->front != NULL) {

/\*处理队列不为空\*/

nexthandelTime = handel\_queue->front->data.handel\_time + nexthandelTime;

total = total + handel\_queue->front->data.handel\_time;

}

}

}

}

}

Status Leave\_event(ElemType \*e ,int i) {

/\*用来处理客户离开事件\*/

event \*p = (event\*)malloc(sizeof(event));

p->index = 0;

p->No = e->No;

p->time = i;

InsertLink(event\_link, \*p);

return OK;

}

double average\_StayTime(LinkList e,double final) {

/\*通过事件列表来计算用户平均逗留时间\*/

double sum=0, a, b;

int shumu=0;

LNode \*p = e->head;

for (int i = 1; i <= e->len; i++) {

while(p!=NULL) {

if (p->data.No == i) {

if (p->data.index == 1) {

a = p->data.time;

shumu = shumu + 1;

}

if (p->data.index == 0) b = p->data.time;

}

p = p->next;

}

p = e->head;

if (b == -1.0) b = final;

sum = sum + b - a;

b = -1.0;

}

printf("全部用户数: %d \n总共消耗时间: %f \n", shumu, sum);

printf("客户平均逗留时间： %f 分", sum / shumu);

return sum / shumu;

}

主函数设计：

int main(void) {

leave\_kehu = (ElemType\*)malloc(sizeof(ElemType));

search\_kehu = (ElemType\*)malloc(sizeof(ElemType));

handel\_queue = InitQueue();

wait\_queue = InitQueue();

event\_link = InitLink();

srand((unsigned int)time(NULL));

int MAXTIME, MAX\_TIMEJIANGE, MAX\_TIMEJIAOYI;

printf("请输入银行的运营时间（单位：分钟）：");

scanf\_s("%d", &MAXTIME);

printf("请输入银行的运营金额（单位：元）：");

scanf\_s("%d", &total);

printf("请输入用户最大交易金额（单位：元）：");

scanf\_s("%d", &max\_money);

printf("请输入俩个到达事件的最大时间间隔：（单位：分钟）：");

scanf\_s("%d", &MAX\_TIMEJIANGE);

printf("请输入用户最大交易时间：（单位：分钟）：");

scanf\_s("%d", &MAX\_TIMEJIAOYI);

//ElemType \*newKehu = (ElemType\*)malloc(sizeof(ElemType));

for (int i = 0; i < MAXTIME; i++)

{

Arrial\_event(i, MAX\_TIMEJIANGE, MAX\_TIMEJIAOYI);

handel\_event(i);

Sleep(1000);

}

LinkTraverse(event\_link);

printf("\n");

average\_StayTime(event\_link, MAXTIME-1);

system("pause");

return 1;

}

动态数据类型实现：

ElemType\* mymalloc( int max\_timejiaoyi) {

/\*动态申请客户节点\*/

ElemType \*e;

e = (ElemType\*)malloc(sizeof(ElemType));

if (e == NULL) return ;

e->handel\_time = (rand() % max\_timejiaoyi) + 1;

e->money = (rand() % (2 \* max\_money)) - 2500;

e->No = kehu\_NO;

kehu\_NO++;

return e;

}

Status myfree(ElemType \*e) {

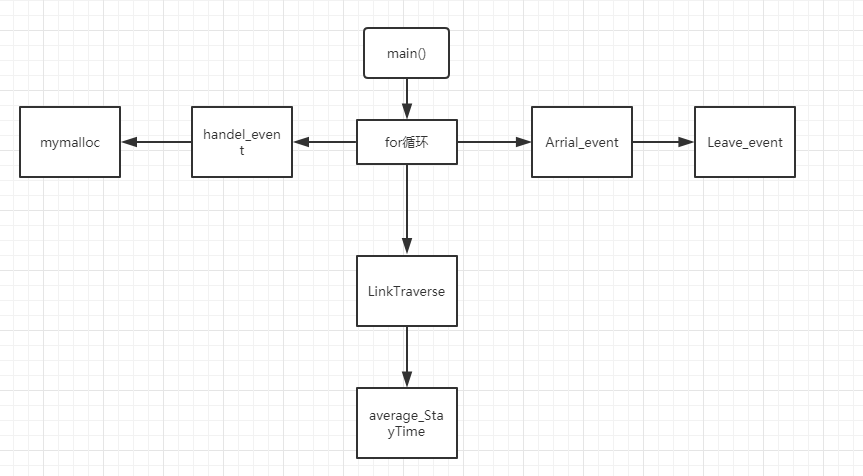
/\*动态释放客户节点\*/

if (e == NULL) return ERROR;

free(e);

return OK;

}



4. 调试分析

1.测试过程中出现一些节点无法访问内存节点的情况，检查动态分配空间操作，找出错误。

2.结果出错，不断整理到达事件、离开事件和队伍操作事件之间的逻辑联系。

3.由于第二个队列的用户可以插在第一个队列用户的对头进行操作，所以增加了合并队列操作，将另外队列合并到一个队列的队头。

4.时间复杂度O（n）

5. 用户使用说明

1.定义链队列抽象数据类型并实现相关操作。

2.定义单链表抽象数据类型并实现相关操作

3.定义银行拥有金额，银行运营时间等全局变量。

3.定义俩个队列（一个处理业务的队列，一个等待队列），一个事件链表用来消除链表

4.设计新用户到达事件操作接口（当当前时间等于用户随机生成的到达时间时触发）

5.设计第一个队列操作接口（当前一个用户业务处理完离开时触发）。

6.设计客户离开事件操作接口（当用户处理完业务时触发）。

使用说明：

运行程序后在控制台输入下面信息（每输入一个信息回车一次）

1.银行拥有的款额

2.银行营业时间

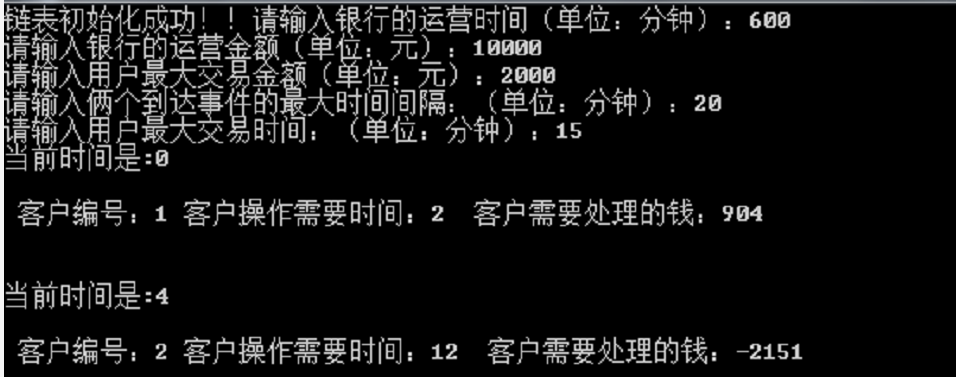
3.用户最大交易金额

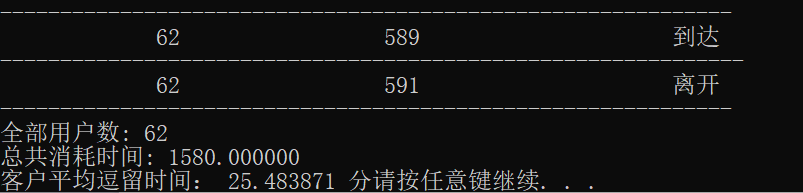
4.俩个用户到达时间的最大值

5.用户最大交易时间

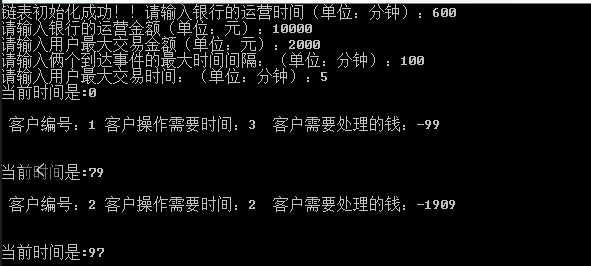
6. 测试结果

第一组数据测试结果：





第二组数据测试结果：





第三组测试结果：

